AMSI VACATION RESEARCH SCHOLARSHIPS 2019–20

EXPLORE THE MATHEMATICAL SCIENCES THIS SUMMER

Advanced Solution Techniques to Multi-Component Optimisation Problems

Duong Thuy Dung Le (Amie)

Supervisors:
Dr. Sergey Polyakovskiy
Dr. Dhananjay Thiruvady

School of Information Technology

Deakin University

Vacation Research Scholarships are funded jointly by the Department of Education and the Australian Mathematical Sciences Institute.

AMSI Australian Mathematical Sciences Institute

Australian Government Department of Education

Contents

Abstract

This project studies a combination of the Travelling Salesman Problem and the Knapsack Problem as a single multi-component optimisation problem. This compound problem, called as Travelling Thief Problem, deals with a set of distinct items, each characterized by its unique profit and weight, distributed among a set of cities on a plane. There is a vehicle that must visit each city once and may collect any item unless the capacity constraint imposed on the vehicle holds. Collecting an item contributes its profit to the total gain, but subtracts a transportation cost relative to its weight that affects the vehicle's velocity. While the velocity is a linear function of the accumulated weight, the transportation cost grows non-linearly. The objective is to find the best possible order of the cities for the vehicle and decide on the best collection of items to pick up so that the total gain is maximised.

To solve the problem, we propose a Branch-and-Check (B&C) algorithm, which is an exact approach inspired from the Logic-Based Bender Decomposition technique. B&C breaks the problem apart as a master and a slave problem. We address the non-linear aspect of the problem in the master problem, which represents it as a piecewise linear approximation and solves it as a mixed-integer program (MIP). The master problem gives a dual bound, which is adjusted by solving the slave dynamic programming problem. B&C solves the slave problem each time it encounters a feasible integer solution as part of the MIP branch-and-bound search until the approach converges to optimality. The experimental results show the effectiveness of our approach compared to the state-of-art exact dynamic programming approach reducing the running time up to two orders of magnitude.

Introduction

The Travelling Thief Problem (TTP) was first introduced by Bonyadi et al. in 2013 [3]. Given a list of cities and a list of items distributed amongst the cities, the thief (which we herein consider as a vehicle) with a knapsack needs to visit each city once and decide on collecting items located in the city. Each item has its own profit and weight. The knapsack has a certain capacity that the total weight of selected items must not exceed. Each collected item contributes its profit to the total profit, but produces transportation cost relative to its weight. The goal is to maximize the total gain calculated by subtracting the transportation cost from the total profit.

TTP, as a combination of two well-known combinatorial optimisation problems, namely the Travelling Salesman Problem (TSP) and the Knapsack Problem (KP), is a recent hot topic in the optimization academic community. The two sub-problems are interdependent since the total weight of selected items incurs transportation cost relative to the travelling time. As a consequence, the optimal packing solution depends on the order in which cities are visited, and the optimal tour depends on the selected items of the knapsack since the weight slows down the vehicle and

increases the transportation cost. Thus, finding the optimal solution for both KP and TSP in the TTP problem is much more complex than solving each component individually.

The TTP reflects the complexity of a real-world application when multiple NP-hard problems of different natures often occur simultaneously. Not only do these problems combine several combinatorial optimization aspects into a single problem, but they also emanate from the compounded complexity of conflicting issues in numerous areas like logistics, planning and supply chain management. Recently, various algorithms were proposed to address the TTP and most of them are heuristic or evolutionary algorithms.

In this report, Section 2 formulates the TTP. In section 3, we explain the application of piecewise linear approximation in the Benders Decomposition. Section 4 demonstrates our exact Branch-and-Check approach based on mixed-integer Programming (MIP) adopting the piecewise linear approximation technique. Section 5 shows the experimental results and compares our approach to the existing dynamic programming algorithm proposed by Wu et al. in [1]. The conclusion is drawn in Section 6 as a summary of the report and discussion of future improvements.

Problem Statement

The Travelling Thief Problem is formally defined as follows. Given are n cities distributed on a plane such that a distance d_{ij} between any city i and j is known. Each city i holds m_i items, each item k in city i is characterized by a weight w_{ik} and a profit p_{ik}. There are m items in total. There is a vehicle that must visit each city exactly once and may collect any item located in the city. The vehicle's capacity is limited to C and its speed remains in the range $[v_{min}, v_{max}]$. Each unit time of travelling costs R that represents a renting rate. The objective of TTP is to maximize the gain $\mathrm{Z}(\Pi, \mathrm{Y})$ by finding an optimal permutation of cities and the corresponding packing plan represented by a bitstring $(y_{21}, \dots, y_{2m_2}, \dots, y_{nm_n})$. $\mathrm{Z}(\Pi, \mathrm{Y})$ is defined as follows:

$$\mathrm{Z}(\Pi, \mathrm{Y}) = \sum_{i=1}^{n} \sum_{k=1}^{m_i} p_{ik} y_{ik} - R \sum_{i=1}^{n} \frac{d_{x_i x_{s(i)}}}{V_i} \tag{1}$$

where $s(i)$ is the successor of the city i, which is the next city of x_i in Π. The traveling time between city x_i and the next city $x_{s(i)}$ depends on the distance $d_{x_i x_{s(i)}}$ between them and the speed V_i which is defined as:

$$V_i = v_{max} - \frac{v_{max} - v_{min}}{C} W_i \tag{2}$$

where W_i is the weight of items collected in city x_i. When empty, the vehicle's speed equals v_{max}, and v_{min} when the vehicle is full.

$$W_i = \sum_{j=2}^{i} \sum_{k=1}^{m_j} y_{x_{jk}} . w_{x_{jk}} \tag{3}$$

The collected weight in the last city x_n must be equal to or smaller than the capacity C, i.e:

$$W_{x_n} \leq C \tag{4}$$

Piecewise Linear Approximation

The unit speed of the vehicle is computed by dividing the unit distance by the travelling time, hence expresses the non-linear relationship between speed and time. The time-velocity formulation is stated as $(v) = 1/v$, where $t(v)$ is the time it takes the vehicle to travel one distance unit along the link $(i\,, i+1)$, for any $i = 1 \ldots n$.

To deal with the nonlinear function, we adopt the piecewise linear approximation technique by representing the curve of the function $t(v)$ as a collection of many linear segments. Our goal is to underestimate the traveling time $t(v)$ for every link $(i\,,\ i+1)$, $i = 1 \ldots n$. We draw linear segments to represent a lower bound, thus approximate the time-velocity function as follows:

(i) The interval $[t_{min}\,, t_{max}]$ is partitioned into λ equal-sized sub-intervals.
(ii) Every sub-interval is projected into the curve of the function $t(v)$ to determine the set of breakpoints A_i.
(iii) For each point $a \in A_i$, the line tangent to the curve of the function is drawn as depicted in Figure 1.

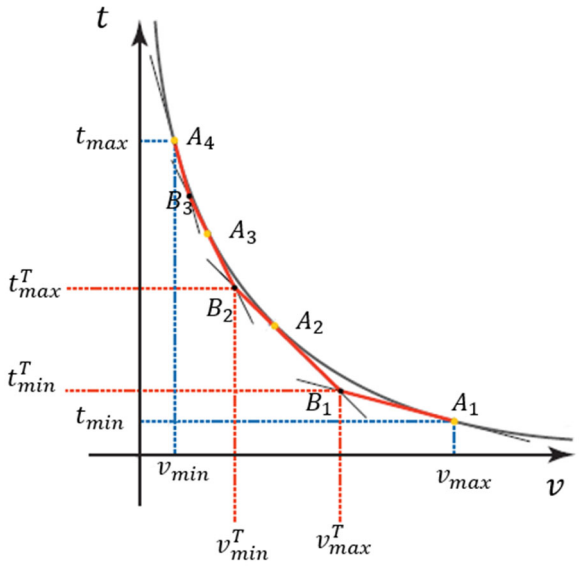

Figure 1: Piecewise linear approximation of $t(v) = 1/v$

Subsequently, the set of breakpoints B_i is defined by the list of intersection points of each pair of neighbouring tangents. This yields a set T_i of $|A_i|$ linear segments produced from a total of $|A_i + 1|$ points. Specifically, (v_{min}^T, t_{max}^T) and (v_{max}^T, t_{min}^T) are the end-points of segment referred to as breakpoints B_i . We can control the precision of the lower bound by adjusting the value of λ. The higher value of λ, the more accurate the curve is approximated. To solve the TTP via mixed-integer programming, the approximated value for each $t_i \in [t_{min}, t_{max}]$ is calculated as the linear combination of two nearest breakpoints $b_i, b_{i+1} \in B_i$, if $v_i \in [v_{min}, v_{max}]$. In another words, if $v_i \in [v_{min}^T, v_{max}^T]$, the approximated value for t_i is equal to the linear combination of t_{min}^T and t_{max}^T. Though there is likely to be a gap between the approximate and the exact result, it is often tight and obtainable in a reasonable time.

The Exact Approach – Branch and Check

Our approach is a Branch-and-Check (B&C) – a form of the Logic-based Benders decomposition (LBBD) [2]. It breaks the solution into a master problem (MP) and sub-problems. The MP is obtained by relaxing the complicating constraints of the original problem. For each optimal solution of the MP, the solution is kept fixed. Subsequently, the slave problem solves the sub-problems to generate Benders cut(s) for the MP. When a feasible solution to the sub-problems is found, it is regarded as a new incumbent solution. The MP is then augmented with new Benders cuts and the process is repeated until the approach converges to optimality. The incumbent is proved to be the optimum solution to the problem.

The difference between LBBD and B&C approach is the time when sub-problems are solved. While in LBBD sub-problems are solved when it achieves the optimum solution for the MP, the B&C solves the sub-problems whenever the feasible solution of MP is obtained. Since the B&C delays the process of finding and proving the optimum solution for the MP, which is time-consuming, it is suitable for the problem where the MP is more difficult than the sub-problems. In the following section, we explain how the B&C works for the TTP.

The time-velocity constraint is considered as the complicating constraint. The TTP divides the solution approach into two parts:

- The master problem (MP) addresses the non-linear aspect of the time-velocity constraint, solves it as a mixed-integer program and provides a dual bound for the TTP.
- The slave problem adjusts the value of dual bound by solving dynamic programing and creates Benders cuts to drive the branch-and-bound search towards an optimal solution.

The MP relaxes the complicating constraint by modelling them as a piecewise linear approximation. As the approximation underestimates the traveling time, the final gain is overestimated. The MP solves MIP model to find the order of the cities (i.e. a feasible tour) and the collection of selected items for each city. Every time the MP encounters a feasible solution, the adjustment of dual bound is delegated to the slave problem. The slave problem iterates as follows:

(i) Given the tour from the MP, it treats the sub-problem as Packing While Travelling (PWT) problem which is solved via dynamic programming as proposed in [8]. The dynamic program gives an optimal packing solution for the given tour.

(ii) It evaluates exact solution provided by the dynamic program and generates extra new constrains for MP:
- If the dynamic program achieves a better result than the incumbent's value, the incumbent is updated, the gap between the dynamic program's solution and the new incumbent is reflected in the MP, and the optimum packing plan is enforced for every possible solution with the same tour structure.
- If the dynamic program fails to obtain a better result than the incumbent's value, the tour is eliminated from the solution space.

Master Problem

The Master Problem is modelled as a mixed-integer program. Let x_{ij} denote a binary decision variable such that $x_{ij} = 1$ if the connection between city i and city j is in use, otherwise $x_{ij} = 0$. The tour is found when all the values of x variables are defined. Let v_{ij} be non-negative decision real-valued variable representing the velocity of the vehicle while traveling from city i to city j and be upper-bounded by v_{max}. Let t_{ij} be non-negative decision real-valued variable representing the traveling time from city i to city j and be upper-bounded by t_{max}. Let y_{jk} denote a binary decision variable such that $y_{jk} = 1$ if item k in city i is selected, otherwise $y_{jk} = 0$. Let $u_i$denote the index of city i in the tour. Let B_i denote all the breakpoints that are generated from linear segments T_i. Then $z_{ib} \in [0,1]$ indicate the weight assigned to the point $b \in B_i$ associated with the pair (t_i, v_i) in the linear combination used to approximate the traveling cost. Finally, let $S \in \mathcal{R}^+$ define the slack variable representing the gap between the objective O and the valid best possible gain determined by the solution of the slave problem for the found tour. We calculate $\mu = (v_{max} - v_{min})/W$ as a constant defined by the input parameters. Then the MIP model built on these variables is defined as follows:

$$max\ O = \sum_{i \in N}\left(\sum_{k \in I_k} p_{ik} - r\sum_{\substack{j \in N \\ i \neq j}} d_{ij}t_{ij}\right) - S \quad (5)$$

$$\sum_{i \in N} x_{ij} = 1 \qquad \forall j \in N \ (6)$$

$$\sum_{j \in N} x_{ij} = 1 \qquad \forall i \in N \ (7)$$

$$v_{ij} \leq v_{max}\, x_{ij} \qquad \forall i \in N, j \in N, \text{ i} \neq j \ (8)$$

$$t_{ij} \leq t_{max}\, x_{ij} \qquad \forall i \in N, j \in N, \text{ i} \neq j \ (9)$$

$$\Omega_i = \sum_{\substack{j \in N \\ i \neq j}} v_{ij} \qquad \forall i \in N \quad (10)$$

$$\Omega_i = \sum_{\substack{j \in N \\ i \neq j}} v_{ij} - \mu \sum_{k \in I_j} w_{jk}\, y_{jk} \qquad \forall j \in N \backslash \{1\} \quad (11)$$

$$\Omega_1 = v_{max} - \mu \sum_{k \in I_1} w_{1k}\, y_{1k} \qquad (12)$$

$$\Omega_i = \sum_{b \in B_i} v^T z_{ib} \qquad \forall i \in N \quad (13)$$

$$\sum_{b \in B_i} z_{ib} = 1 \qquad \forall i \in N \quad (14)$$

$$\sum_{b \in B_i} t^T z_{ib} \leq \sum_{\substack{j \in N \\ i \neq j}} t_{ij} \qquad \forall i \in N \quad (15)$$

$$u_i - u_j + (n-1)x_{ij} \leq n - 2 \qquad \forall i \in N, \forall j \in N, i \neq j \quad (16)$$

Eq. (5) defines the objective function, which maximizes the final gain. Eq. (6) allows exactly one incoming city to city j. Eq. (7) allows exactly one outgoing city from city i. Eq. (8) limits the value of velocity v_{ij} bounded by v_{max}; if the path from city i to city j is not in use, $v_{ij} = 0$. Eq. (9) limits the value of time t_{ij} bounded by t_{max}; if the path from city i to city j is not in use, $t_{ij} = 0$. Eq. (10) gives the velocity of the vehicle when it arrives in city i by summing up all incoming velocity values of other cities. Eq. (11) gives the velocity of the vehicle when it departs from city i by summing up all incoming velocity values then subtracting velocity produced by picking up items from city i. Eq. (12) is a special case of Eq. (11) which determines the velocity value of the vehicle when it departs from the first city. Eq. (13) calculates the speed as a linear combination of two closest breakpoints. Eq. (14) forces the total weight of all breakpoints to be exactly 1. Eq. (15) bounds (approximates) the traveling time. Eq. (16) introduces the sub-tour elimination constraint to avoid occurrence of multiple sub-tous.

Slave Problem

Every time the MP obtains a feasible integer solution, the slave problem is called to evaluate the solution from MP and generate Benders Cut(s). The slave problem solves the PWT problem: Given the input as a tour from MP and the set of items, the dynamic program [8] returns the optimal packing solution and the exact total gain. Let O^{best} be the incumbent solution initialized to $-\infty$. Let O^{master} represent the value of feasible solution obtained from the MP. Let O^{slave} be the exact value and Y a collection of cities representing the most profitable packing plan produced by the dynamic program for a given tour. The two possible scenarios exist.

(i) If the dynamic programming's solution achieves a better result for the problem, i.e. $O^{master} > O^{best}$ and $O^{slave} > O^{best}$, the MIP model adds three Benders cuts, as follow:

$$\sum_{i=1}^{n} x_{[i][i+1]} - \frac{1}{|Y|}\sum_{i\in N}\sum_{k\in I_k} y_{ik} \leq n-1 \quad (17)$$

$$(O^{master} - O^{slave}) - \mathrm{M}.\mathrm{n} \leq S - \mathrm{M}.\sum_{i=1}^{n} x_{[i][i+1]} \quad (18)$$

$$O \geq O^{slave} \quad (19)$$

where $|Y|$ defines the number of items selected in packing plan Y. Let M be a number which is big enough to prevent getting infeasible solutions. Note that when the same tour appears again, $\sum_{i=1}^{n} x_{[i][i+1]} = n$. Eq. (17) forces the program to obtain the best possible packing plan Y for every possible solution with the same tour structure. In other words, whenever the MP finds the same tour again, the packing plan Y is chosen without requesting the MP to continue running to find a new packing plan. Eq. (18) sets the slack variable S to the new value$(O^{master} - O^{slave})$ if the same tour structure occurs. Eq. (19) increases the lower bound of the objective solution to the new incumbent value.

(ii) If the dynamic programming fails to obtain a better result for the problem, i.e. $O^{master} > O^{best}$ and $O^{slave} > O^{best}$, the MIP introduces a single Benders cut, as follows:

$$\sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{x}_{[\mathrm{i}][\mathrm{i}+1]} \leq \mathrm{n}-1 \quad (20)$$

This forces the tour to drop at least one of the selected edges in the undesired solution. In other words, this is to exclude that tour from further consideration.

Computational Experiments and Results

In this section, we compare the performance of our Benders B&C approach with the existing dynamic-programming (DP) approach in terms of the running time influenced by increasing the number of cities and the number of items. The investigation of the computational complexity is performed in a twofold manner: (i) increasing the number of cities, and (ii) increasing the number of items when keeping the fixed number of cities. We run the experiments using small instances following the benchmark proposed by Polyakovskiy et al (2014). The three knapsack types defined from the benchmark are: (i) bounded strongly correlated type: w_{ik}~p_{ik}; (ii) uncorrelated with similar weights: items have similar weights type w_{ik} but different p_{ik}; and (iii) uncorrelated type: w_{ik} and p_{ik} are uniformly distributed.

Table 1 shows the computation results of DP and B&C for small-sized instances. Columns *n* and *m* indicate the number of cities and the number of items, respectively. Column OPT shows the optimal

solution of the final gain for the instances. Column DP – RT gives the running time for the DP approach whereas column B&C – RT represents B&C's running time.

Table 1. Running time of DP and B&C approach.

Knapsack Type	**n**	**m**	**OPT**	**DP – RT (in seconds)**	**B&C – RT (in seconds)**
bounded strongly correlated type	10	9	573.897	1.21	0.84
	10	45	1091.127	14.89	0.56
	12	11	648.546	4.58	2.70
	12	55	1251.780	117.99	1.69
	15	14	547.419	39.82	3.64
	15	70	920.372	3984.29	2.38
	16	15	794.745	105.5	18.21
	17	16	685.565	248.6	31.08
	20	19	518.189	4533.7	129.32
uncorrelated type	10	9	1125.715	0.93	0.86
	10	45	6009.431	6.39	1.46
	12	11	1717.699	3.94	1.61
	12	55	8838.012	35.79	3.87
	15	14	2392.996	89.46	12.37
	15	70	9922.137	740.22	33.32
	16	15	2490.889	59.5	25.66
	17	16	2342.664	134.9	29.61
	20	19	2092.673	2456.9	967.95
uncorrelated with similar weights	10	9	753.230	0.86	0.61
	10	45	3009.553	8.87	0.66
	12	11	774.107	3.36	1.06
	12	55	3734.895	38.36	20.16
	15	14	637.419	16.35	1.60
	15	70	4659.623	867.78	4.12
	16	15	540.897	36.4	1.91
	17	16	556.851	70.8	2.85
	20	19	451.052	1007.7	131.52

The B&C approach shows its effectiveness in solving the TTP problem. The difference between running times of the two approaches can be clearly shown when the number of cities and the number of items increase. The comparison reveals that the B&C can reduce the running time up to two orders of magnitude. This is expected because the search of the MP is guided better from the specificity that is drawn from Benders cut(s) derived by the slave problem. Thus, the MP prevents the investigation of large infeasible solutions, specifically those having a large number of cities and items. In the B&C approach, the optimum packing plan, once computed for a tour, is propagated to

all possible solutions with the same tour component. Therefore, it avoids the searching for undesirable solutions that are clearly worse than one obtained. This leads to a better performance in solving the large instances.

Conclusion

We have introduced the B&C exact approach to address the non-linear aspect of the TTP by representing it as a piecewise linear approximation and solving it as MIP. The B&C approach is guided by adding new constraints to the MIP model after each iteration obtains a feasible solution. Therefore, it can significantly reduce computation time. The experimental results show that small-sized instances can be solved to optimality in the reduced time when compared with the existing dynamic-programming approach.

As for future work, we aim to solve moderate-sized instances, which can be efficiently handled by applying the strengths of evolutionary computing and machine learning. By running some heuristic algorithms to obtain the pool of optimal and near-optimal tours, such as Chained Lin-Kernighan proposed by Applegate et al. [7], we aim to use the pool to explore higher-order interactions between the decision variables. Some approaches that could be used for learning process are based on Linkage Tree Model proposed by Thierens et al (2018) [6]. The heuristic and statistical information obtained from the analysis then can generate constraints to warm-start the branch-and-check search. By making the B&C a heuristic algorithm, we may lose some good solutions. Nevertheless, we expect the new heuristic learning approach to guide the branch-and-bound search for the master problem towards the most promising parts of solution space.

References

[1] Wu J, Wagner M, Polyakovskiy S & Neumann F 2017, 'Exact Approaches for the Travelling Thief Problem', Asia-Pacic Conference on Simulated Evolution and Learning, Springer, pp. 110-121.

[2] Hooker J & Ottosson G 2003, 'Logic-based benders decomposition', Mathematical Programming 96, pp. 33-60.

[3] Bonyadi M R, Michalewicz Z & Barone L 2013, 'The travelling thief problem: the first step in the transition from theoretical problems to realistic problems', Evolutionary Computation (CEC), 2013 IEEE Congress, pp. 1037-1044.

[4] Polyakovskiy S & Neumann F 2015, 'Packing while traveling: Mixed integer programming for a class of nonlinear knapsack problems", International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, Springer 2015, pp. 332-346.

[5] Polyakovskiy S, Bonyadi M R, Wagner M, Michalewicz Z & Neumann F 2014, 'A comprehensive benchmark set and heuristics for the traveling thief problem, Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation, ACM, pp. 477–484.

[6] Thierens D, Bosman P (2018): Model-based evolutionary algorithms: GECCO 2018 tutorial. In Proceeding GECCO '18 Proceedings of the Genetic and Evolutionary Computation Conference Companion: ACM, 553-583.

[7] Applegate D, Cook W J & Rohe A 2003, 'Chained Lin-Kernighan for large traveling salesman problems', INFORMS Journal on Computing, 15 (1), pp. 82–92.

[8] Neumann F, Polyakovskiy S, Skutella M, Stougie L, Wu J 2019, 'A Fully Polynomial Time Approximation Scheme for Packing While Traveling', Algorithmic Aspects of Cloud Computing, ALGOCLOUD 2018, Lecture Notes in Computer Science, vol. 11409. Springer, Cham.